Decoder Optimization Plug-in for ViPro User Guide

Written by: Domenic Carr (EE – UVA Class of 2011)
May 2011

Introduction

This document explains how to use the decoder optimization program. It provides examples of how to use the program and offers methods to adjust the program in the future to analyze different decoder structures. For a detailed explanation of the theory behind the program, the limitations of the program, and the results of a static decoder structure, please refer to the document "Improving the Performance of a CAD Tool Designed to Generate Optimized Virtual Prototypes of Complete SRAM Macros". This document is on the UVA ECE Wiki at the following link: http://venividiwiki.ee.virginia.edu/twiki/bin/view/Main/ViPro.

Getting Started

To run this program, you must be logged into the ECE Linux Servers (either ivycreek.ece.virginia.edu or blackrock.ece.virginia.edu). For detailed instructions on how to login to the Linux Servers, please refer to the appropriate tutorial on the UVA ECE Wiki. To start using this program, download the zip file SweepRows.zip from the ViPro page on the UVA ECE Wiki (http://venividiwiki.ee.virginia.edu/twiki/bin/view/Main/ViPro) to any directory in your Linux environment, then unzip the file.

Once you have downloaded the file and unzipped all of its contents into a directory, open Konsole (terminal window) and start Cadence. For help starting Cadence, please refer to the appropriate tutorial on the UVA ECE Wiki. The way I started Cadence was by running the following commands:

". cadence-mmsim"

". calibre-setup-new"

"tcsh"

"source setup-cadence"

Notice that there is a period that starts the command. Also notice that there is a space between the period and the first letter of cadence/calibre

Obviously, running these commands requires that you have the necessary files (e.g. cadence-mmsim, calibre*, setup-cadence) in the directory from which you are executing the commands.

Once you have successfully started Cadence, "cd" to the directory where you unzipped the contents of SweepRows.zip. Once there, "cd" into the SweepRows folder. The entire decoder optimization program is located within this folder.

Contents of the Decoder Optimization Program

This section of the User Guide will explain all of the files and folders used by the Decoder Optimization Program. It will show the contents of each folder and explain the importance of each file within the folders.

SweepRows Folder

The SweepRows folder contains the perl script (SweepRows.pl) that executes the program, the ocean script (run.ocn) that runs the Cadence simulations on the decoder netlists, two important configuration files (output_format.txt & params) as well as seven important directories (*bits directories). This directory also contains the final output of the optimization program (results.txt). Figure 1 shows what the SweepRows folder should look like.

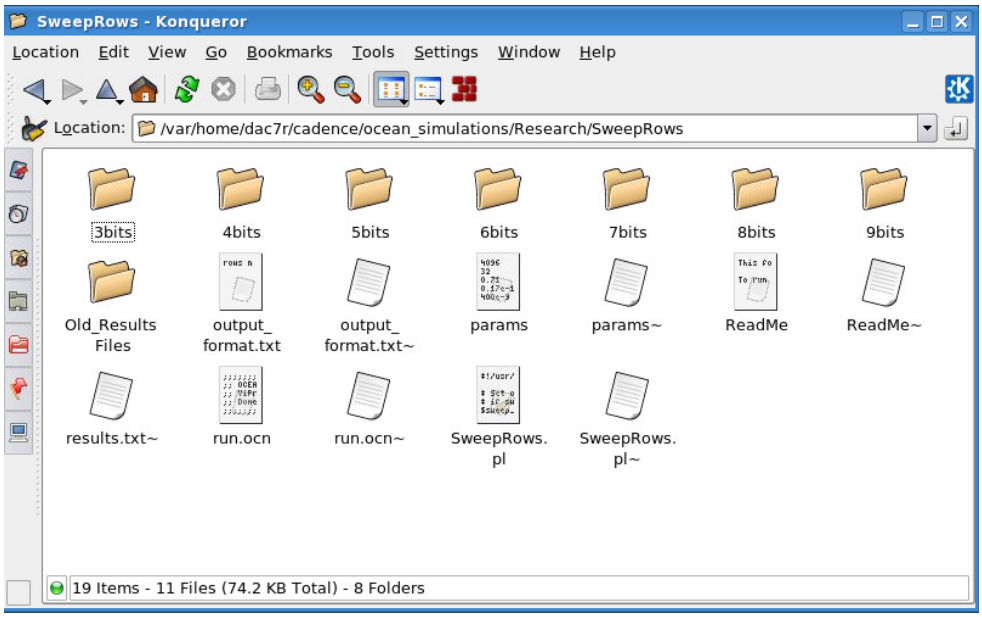

Figure 1: SweepRows directory

SweepRows.pl

SweepRows.pl is the top level of the program. This file is called from the terminal window to start executing the program. To run the decoder optimization program, be sure you are located in the SweepRows folder, then type "perl SweepRows.pl" into the terminal window. Currently, this command takes in no command-line arguments. Executing this command will run the program with all of the default parameters and decoder netlists. The User Guide will detail how to change the default parameters and netlists in a later section.

SweepRows.pl allows you to test one of the three things by adjusting certain flags at the top of the file. The three possible tests you can run are:

1) Sweep across all possible combinations of n and k buffers for all possible row/column configurations.
2) Sweep across all possible combinations of n and k buffers for one row/column configuration.

3) Select one combination of n and k buffers for a particular row/column configuration

Currently, you have to choose which test you want to run before executing the "perl SweepRows.pl". To do this, you have to open the SweepRows.pl file, and adjust the flags at the top of the file. To run Test 1, the $sweep_flag variable should be set to 1. All other flag variables are ignored. An example of what the top of the file should look like for sweeping across all n and k buffers and all possible row/column combinations can be seen in Figure 2.

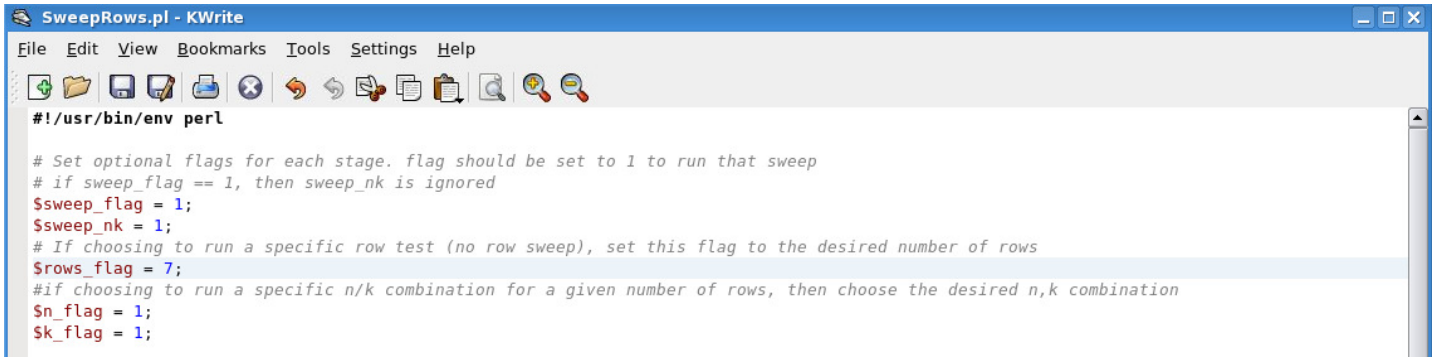

Figure 2: Example of how to run Test 1

To run Test 2, the $sweep_flag variable should be set to 0 and the $sweep_nk variable should be set to 1. Then, choose the desired number of row address bits by setting the $rows_flag equal to the number of row address bits you want to test (possible values are the integers 3 through 9). The $n_flag and $k_flag variables will be ignored. An example of what the top of the file should look like for sweeping across all n-k buffer combinations for a 256 row memory (8 row address bits) can be seen in Figure 3.

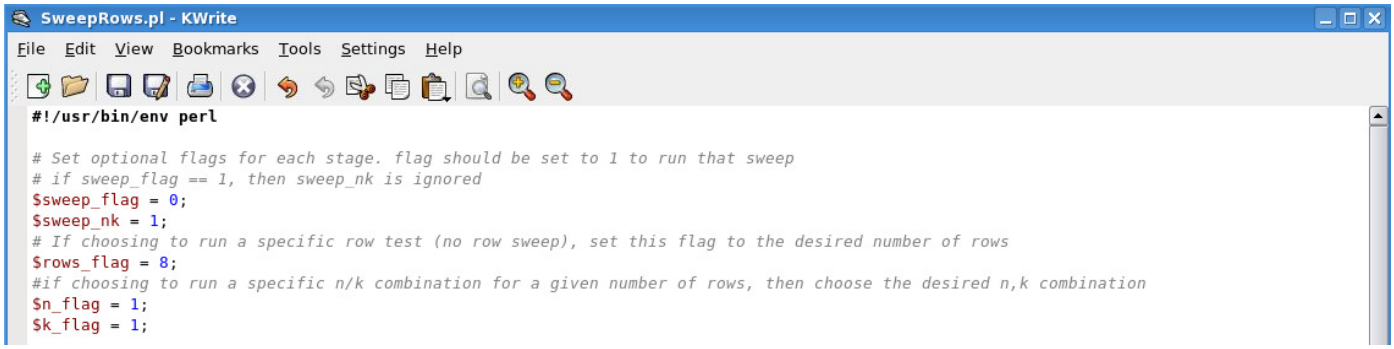

Figure 3: Example of how to run Test 2

To run Test 3, the $sweep_flag variable should be set to 0 and the $sweep_nk variable should be set to 0. The $rows_flag variable should be set to the desired number of row address bits. The $n_flag variable should be set to the desired number of n-buffers (possible values are 0, 1 and 2). The $k_flag variable should be set to the desired number of k-buffers (possible values are 0, 1, and 2). An example of what the top of the file should look like for selecting a 64 row memory (6 row address bits) with 1 n-buffer and 2 k-buffers can be seen in Figure 4.

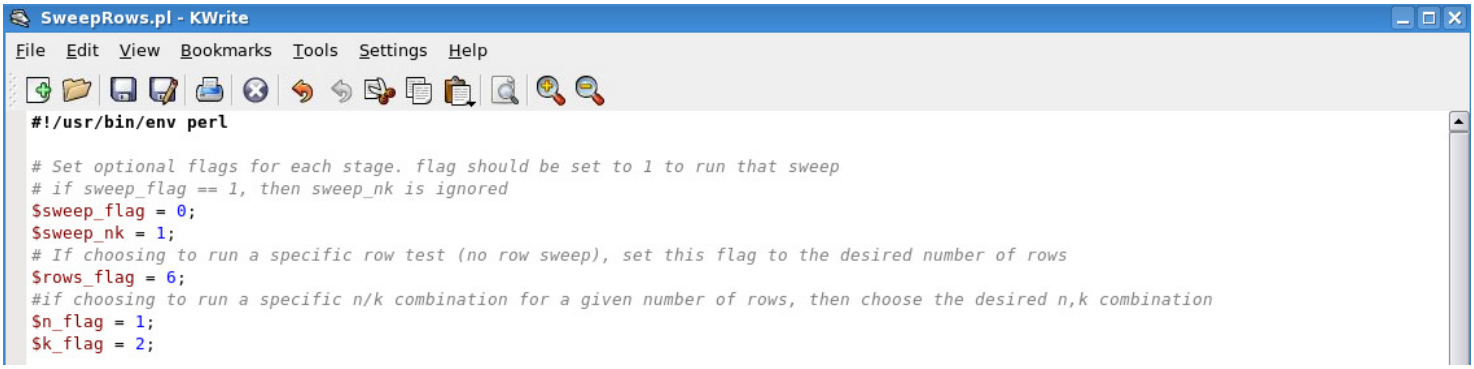

Figure 4: Example of how to run Test 3

In the future, I envision being able to select which test you want to run at run-time, instead of having to adjust the flags manually. This can be achieved by setting the values of the flag variables to the values of command-line arguments.

Params

The params file contains global variables for the memory. These global variables are constants across all row/column combinations, and are therefore stored in the "upper-level" of the program. These variables are used by the run.ocn file when it is simulating the netlists in Cadence. The current params file can be seen in Figure 5. Currently, there are only 5 global variables: memory capacity, size of word, parasitic resistance of wire per micron, parasitic capacitance of wire per micron, and height of bitcell. To change any of these values, simply

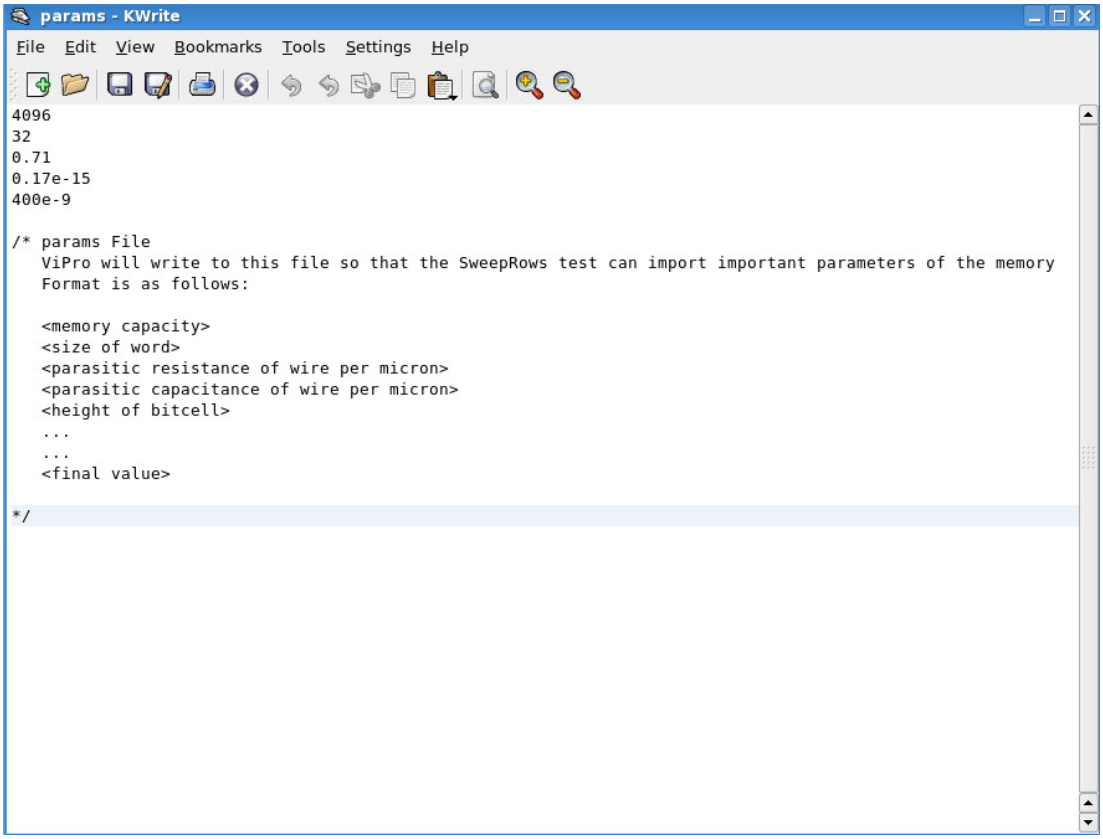

Figure 5: params file

open the params file, adjust to the desired values, save, then close the file. Values for memory capacity and word sizes should be powers of 2. Based on ViPro's current architecture, there are limits to the minimum and maximum sizes of the memory capacity. Thus, values for the memory capacity fall within a certain range, which is based on the word size. If you choose a value out of the range, though, the program will not crash. It will simply throw an error at runtime that your memory capacity is out of range and stop the simulation. To understand how to calculate the possible range of values for the memory capacity for a given word size, please consult the "Improving the Performance of a CAD Tool Designed to Generate Optimized Virtual Prototypes of Complete SRAM Macros" document.

output_format.txt

The configuration file output_format.txt is used by SweepRows.pl to put a header on the output of the decoder optimization program. SweepRows.pl uses the "cat" command to put the contents of output_format.txt into results.txt. Thus, the header seen in the results.txt file will be whatever is in the output_Format.txt file. A diagram of this relationship is in Figure 6. This file should be coordinated with the print statements found in the run.ocn file (to be explained in depth later). Whatever output values you want the optimization program to report, you should include its label in the header of output_format.txt. Particular spacing and indentation is arbitrary. The current spacing between the words in the header was chosen to look most aesthetically pleasing with the output requested from the run.ocn file.

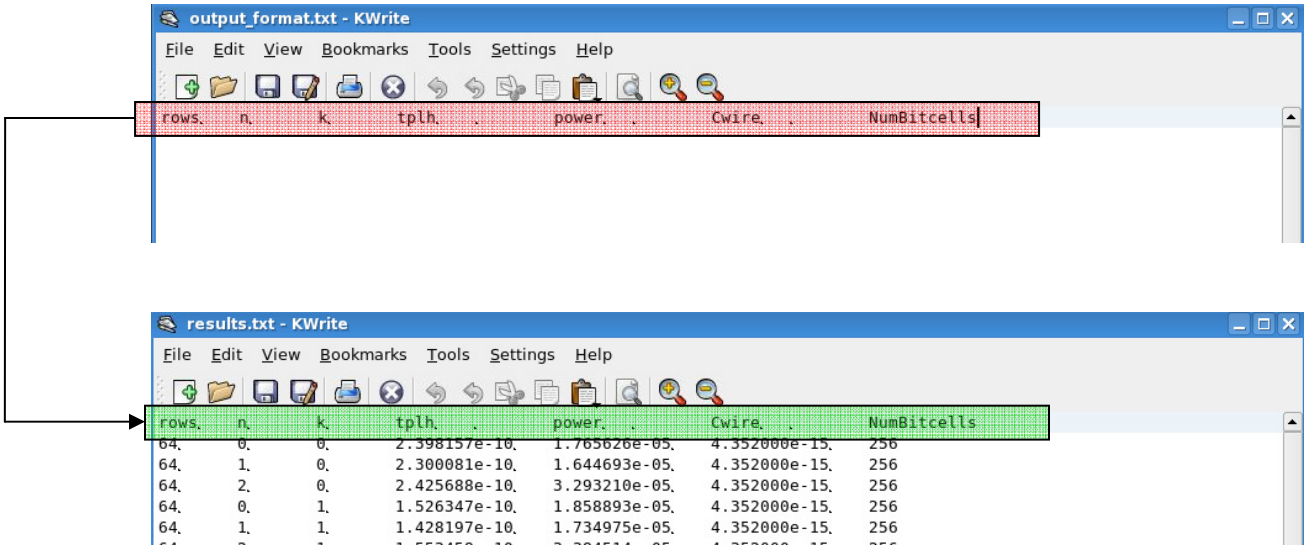

Figure 6: output_format.txt contents becoming header of results.txt

run.ocn

The run.ocn file is the ocean file that controls the Cadence software to perform simulations on the different decoder netlists found in the *bits directories. This run.ocn file is a "generic" ocean file that the SweepRows.pl file copies into each *bits directory so that when

ocean is executed, it runs this file. As a "generic" file, this ocean file cannot run as a standalone in directory by itself (i.e. you can't start ocean on your own and execute the command "load run.ocn"). It requires several configuration files at the "row-specific" level, which provide row-specific information to the ocean file at runtime.

It is important to note, however, that using this run.ocn file as the only ocean file is a key strength of this program. For instance, instead of having multiple run.ocn files and having to change the same line of code in seven different places when you want to make a change to the simulation procedure, you only have to make the change in one spot. The trade-offs for this are the added overhead of creating configuration files and executing more computer instructions/operations to copy run.ocn into *bits directory, then deleting it after use.

A detailed explanation of the run.ocn file can be found at the end of the User Guide, after the remaining contents of the program are explained.

***bits directories**

These directories are the places where the ocean-specific files can be found (i.e. netlist, netlistFooter, netlistHeader, etc.) as well as non-changing row-specific parameters found in the bitspec file. These directories are named 3bits, 4bits, 5bits, 6bits, 7bits, 8bits, and 9bits. The directory names have to be spelled exactly like this (i.e. no space, case-sensitive, etc). The reason for this requirement is because the SweepRows.pl file uses these names to iterate through the different row-bit directories from the top level of the program. Since "bits" is common to each of the directory names while the number preceding it changes, the perl script initializes a variable to hold the value "bits", then it iterates through the directories by concatenating a different integer (3 through 9) to the front of it (thus creating 3bits, for example). If you are

interested in changing the names of these directories, it is recommended to follow a pattern such as this (keep part of the name constant and just change a number at the beginning or end of the string) for easy manipulation in the SweepRows.pl file.

6bits

Since each *bits file contains the same contents, the User Guide will only discuss one of these directories in detail. The same explanation holds for all other *bits directories. The typical *bits directory looks like that which is seen in Figure 7. For the decoder optimization procedure

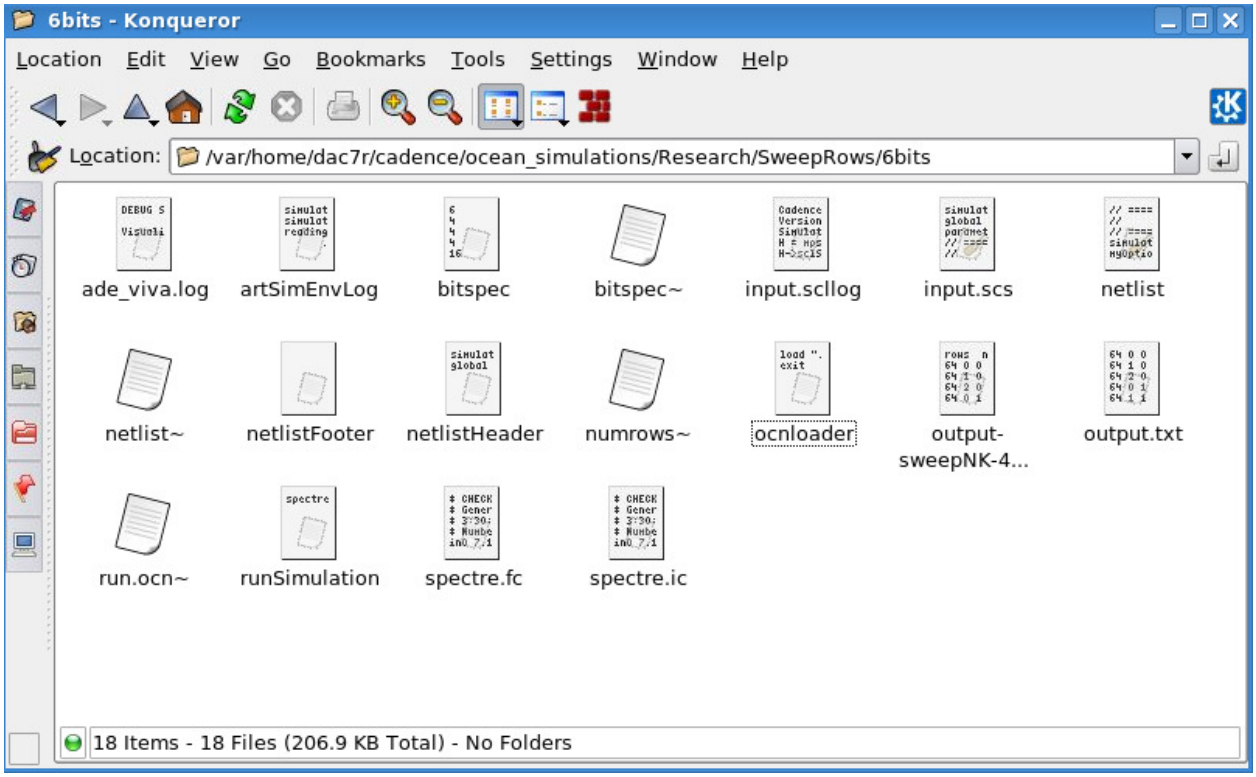

Figure 7: Example *bits directory (6bits directory shown)

to work, there are five required files. These required files are (in alphabetical order): bitspec, netlist, netlistFooter, netlistHeader, and ocnloader. The names of these files must be spelled exactly this way. The other files in this folder are by-products of simulations. The only important by-product of simulations is the output.txt file, which is the output of the run.ocn file. After the simulation is complete for the 6bits directory, the contents of output.txt are copied by

the SweepRows.pl file and concatenated into the results.txt file. By concatenating the contents of output.txt files from the other *bits directories, the results.txt file is populated with the full results of the simulation. The file output.txt must also be spelled exactly this way (though this is determined by the run.ocn), since the SweepRows.pl file looks for the filename “output.txt”.

netlistFooter

The file netlistFooter is required for successful simulation in Ocean. It is a blank file. Its filename must be spelled exactly as shown. During simulation, netlistFooter is appended to the end of netlist and is used by the simulator to mark the end of the netlist. For more detailed information about netlistFooter, consult the Ocean tutorial on the wiki.

netlistHeader

The file netlistHeader is required for successful simulation in Ocean. It has two lines of code: on the first line, “simulator lang=spectre”; on the second line, “global 0”. Its filename must be spelled exactly as show. During simulation, netlistHeader is added to the beginning of the netlist

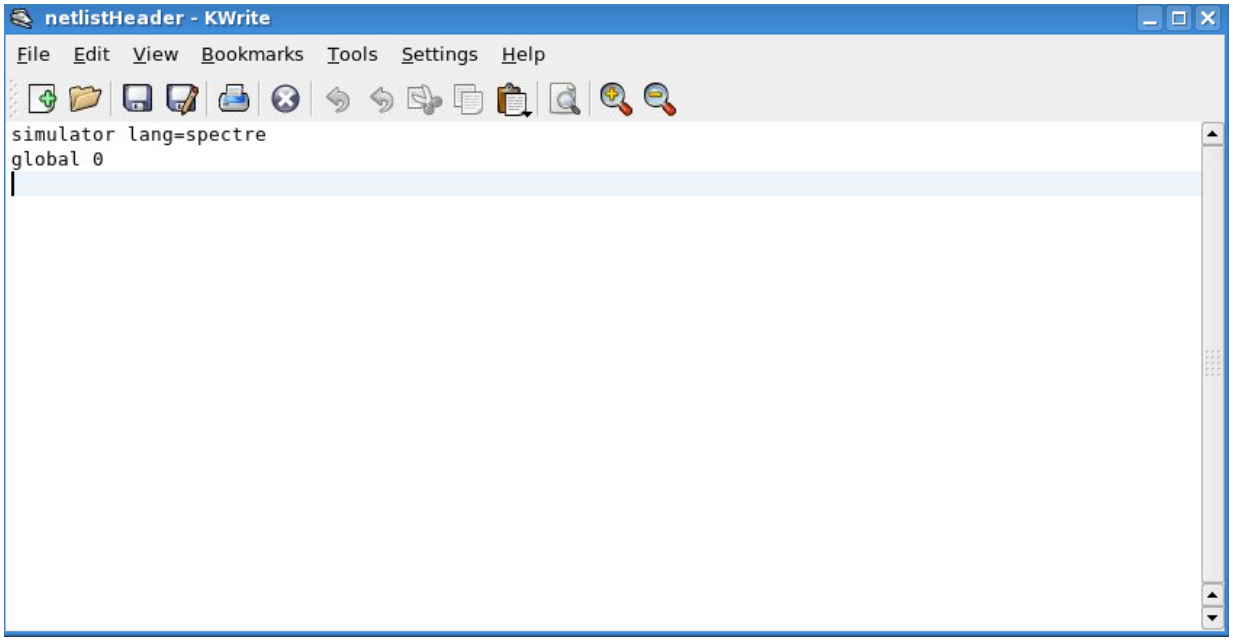

Figure 8: netlistHeader

and relays important information to the simulator. The file netlistHeader can be seen in Figure 8. For more detailed information about netlistHeader, consult the Ocean tutorial on the wiki.

ocnloader

The file ocnloader is required to run an ocean simulation in an automated way from the terminal window. As SweepRows.pl iterates through the different *bits directories, it executes the command "ocean –nograph < ocnloader". This loads the ocnloader script upon successful startup of Ocean, and executes the contained commands. For the decoder optimization program, all ocnloader needs to do is call the run.ocn file using the 'load "./ run.ocn";' command, and then 'exit' upon finishing the simulation of run.ocn. Notice the "./" at the start of the command, which signals the run.ocn file is in the current working directory. This works because the run.ocn file has already been copied into the *bits directory by that point in the SweepRows.pl script. It would be interesting to see if you can cut out the copying of the run.ocn file into the different *bits directories by simply using the command 'load "../run.ocn";' in the ocnloader file. The "../" signifies moving up to the parent directory. The filename must be exactly as shown,

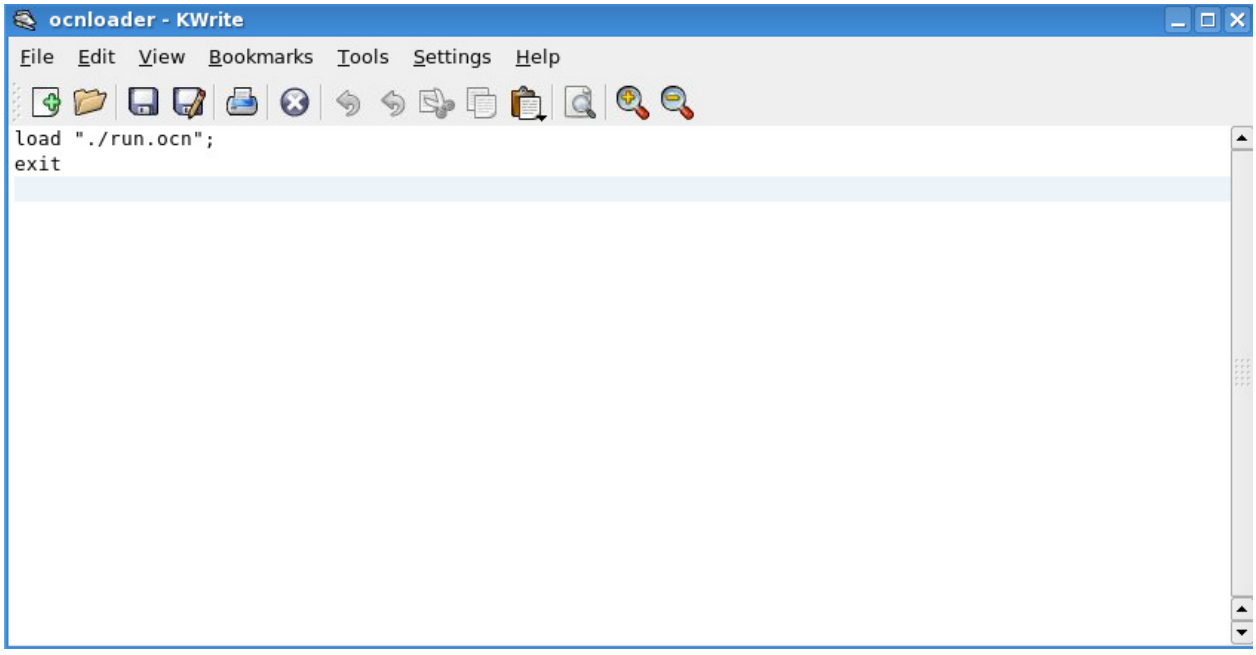

Figure 9: ocnloader

because SweepRows.pl uses this exact name. If you were interested in changing the name of the run.ocn file, this is where you would tell the optimization program that you wanted to do that by simply making the load command call the filename you are interested in simulating. The ocnloader file can be seen in Figure 9.

bitspec

The file bitspec is a configuration file at the "row-specific", or bit-specific, (hence the name bitspec) level. It contains parameters that the run.ocn file uses to convert the ocean script from a generic script to a bit-specific script. bitspec contains 6 parameters run.ocn reads in at runtime. These parameters are different for each *bits directory. The bitspec file for the 6rows directory can be seen in Figure 10. The first value in the file is the number of row address bits. The remaining values are all scalars that are used as multipliers to calculate the energy of the decoder from the worst-case path. These will be explained in more detail in the detailed explanation of the run.ocn file. The format of the file (one value per line) is due to the way the run.ocn file reads in values from a file. The run.ocn file is told to look for one value per line,

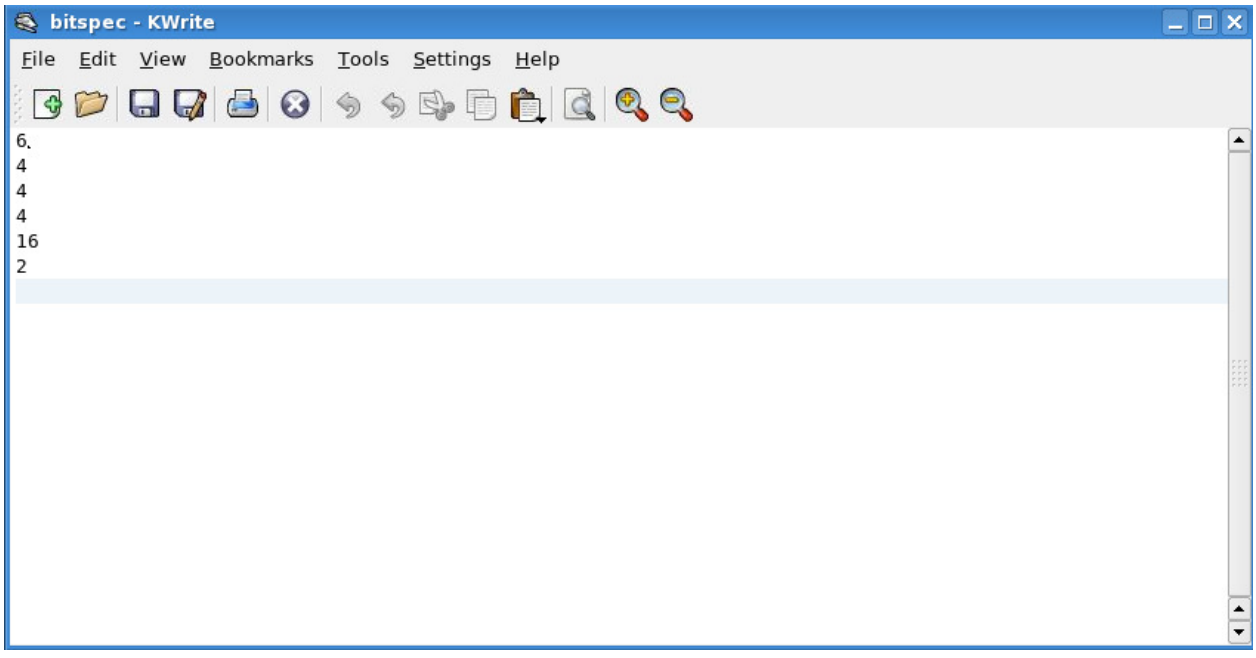

Figure 10: Example of bitspec file (for 6bits directory)

then perform a newline operation, then read in the next value. Thus, the format of bitspec follows this pattern.

netlist

The netlist file is where the circuits used to model the decoder are located. The netlist file is nearly identical across each *bits directory. The only differences are located in the implementation of some of the subcircuits, but the top-level circuit structures are the same. Currently, the model files for the 45 nm freePDK technology are "included" at the top of the netlist. These include statements would be different when using different process technologies.

There are nine models per netlist – one for each n-k combination of buffers. The naming convention for these subcircuits is as follows: "WCpathN<[0,1,2]>K<[0,1,2]>, where you replace the <[0,1,2]> with the integer 0, 1, or 2. For example, the model with 1 n-buffer and 2 k-buffers would have a name of "WCpathN1K2". All nine of these models have the same structure - each model has 8 subcomponents, each of which is named the same across models (Inv0, Predecoder0, Nbuffer0, PAR0, Nand2, Inv2, WLdriver0, Kbuffer0). The reason each of these subcomponents has the same name across models is so that the run.ocn file can calculate the power consumption of the model easily. If the names were different across models, then the run.ocn file would have too many "case" or "if-then" statements everywhere in the code, since the ocean testing language requires you type in the name of the subcircuit whose power consumption you want to measure prior to runtime. Thus, by using the same structure across each model and having each subcircuit have the same name (even if they have different implementations) the run.ocn file can call one set of names for all decoder models to measure the power consumption. An example of this can be seen in Figure 11.

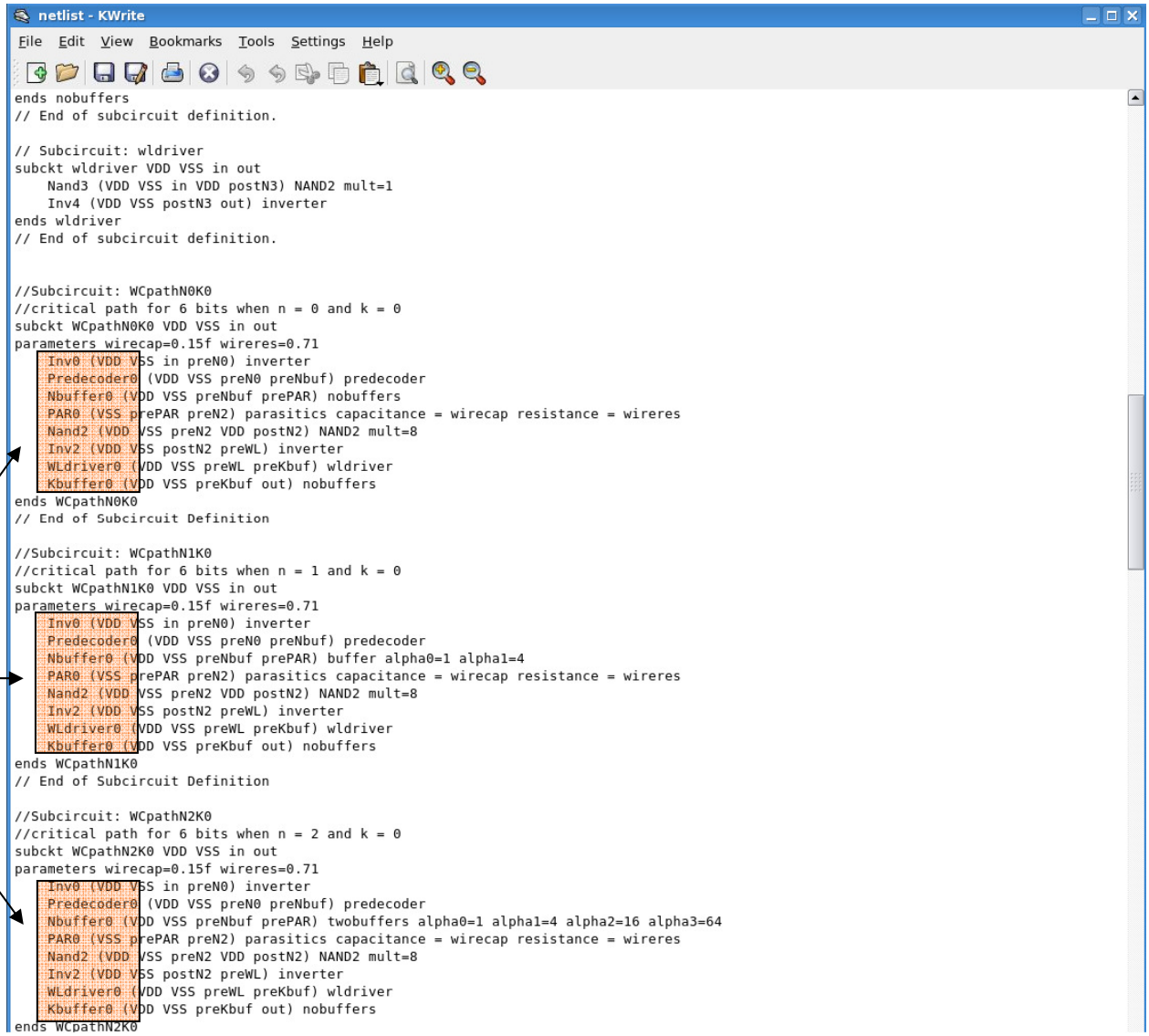

Figure 11: Each Decoder Model Follows Same Structure in Netlist

Each model takes in two parameters – wirecap and wireres. These are the values of parasitic resistance and capacitance of the predecoder wires. These values are generated in the run.ocn script and then are passed to the netlist by using design variables. Then, by assigning those design variables to the parameters taken in by the models, the models will obtain the proper parasitic capacitance and resistance of the predecode wires. As a consequence of every model in a given netlist having the same number of row address bits (meaning each model has

the same amount of rows and thus having the same memory height) every model in a given netlist will have the same predecoder wire resistance and capacitance.

This User Guide will not discuss the intricacies of each of the subcircuits that compose the models in too much detail. However, a few important things should be mentioned. First, each of the subcircuits is parameterized, which allows for a change at the top of the hierarchy to trickle down to the bottom of the hierarchy. For example, this means that to size up a buffer at requires only having to change the alpha parameter at the instantiation of the buffer, since that size parameter is then passed down to all of the subcircuits abstracted beneath it. Second, the nobuffers subcircuit is simply a resistor with a very small resistance (to simulate a near-lossless connection). Third, in some of the predecoder subcircuits, the Nand0 component has "in" being fed to both input terminals of the Nand gate while its mult is 1, instead of "in" being fed into only one input terminal and its mult being 2. This is intentional. This is just a better way to model the transition of that particular NAND gate on the worst-case path since the both inputs of the NAND gate are changing simultaneously.

Aside from the subcircuits that comprise the models (nands, inverters, buffers, etc.), another important feature of the netlist file is that it contains the test bench for the different models. This is done by setting up 9 unique testbeds, one for each model. Each testbed has its own power supply, input signal, buffer chain, device under test (DUT) – the model itself, and the bitcells loading the word line at the output of the DUT. The naming convention for these devices can be seen in Table 1. Suffix refers to the number value after the base name of the testbed component (e.g. V0, VVIN0, B0, etc. all belong to the test bench associated with n=0, k=0). Finally, there is also a one global voltage supply that powers all buffers and bitcells.

Suffix	N	K
0	0	0
1	1	0
2	2	0
3	0	1
4	1	1
5	2	1
6	0	2
7	1	2
8	2	2

Table 1: Mapping Suffix Values to N-/K-Buffer Combinations

Detailed explanation of run.ocn

Now that the other portions of the program have been explained, it is appropriate to revisit the run.ocn file and explain its operation in a little more depth. In essence, the run.ocn file is split up into three sections. The first section deals purely with initialization. The second section deals with reading in the various parameters files. The third section is where the calculations and Cadence simulation actually occur.

In the initialization section, the environment is initialized and variables are initialized. Regarding the simulation environment, the simulator is set to spectre, the netlist file is specified as one that will be in the current working directory, and the results directory is specified (although this directory will later be deleted by SweepRows.pl since it takes up too much hard drive space. Regarding variable initialization, variables that will be needed later in the code are set either to zero or to some reasonable nonzero value. Some of these variables are simply operated on in the code, while others are temporary place holders for variables that will be read in from parameters files. The run.ocn file is well-commented in this section to explain the importance of each variable.

In the parameter files read-in section, three files are read by the run.ocn file. These files are the params file (located in the SweepRows directory), the bitspec file (located in the same *bits directory), and a temp.txt file (will be discussed in detail shortly). When the params file is read in, it assigns values to the variables capacity, size_word, res, cap, and height_bitcell. These variables mean (in same order as just presented), the capacity of the memory being test, the size of a word for this memory, the parasitic resistance per micron of the predecode wire, the parasitic capacitance per micron of the predecode wire, and the height of a bitcell in the particular technology. If you want to adjust the variables that are read in from the params file, simply adjust the fscanf command in the ";;READ INPUT PARAMS FILE" section. Also, notice that the second argument in the fscanf command is the format and location of the values being read in. This should match what is in the params file in order to read the values into the run.ocn file correctly.

When the bitspec file is read in, the variables r, predecoderEnergyMult, nBufferEnergyMult, fanoutEnergyMult, fanoutinvEnergyMult, and wldEnergyMult are all assigned values. For detailed explanations of what each of these variables is, please consult the comments in the run.ocn code. Basically though, any values that are specific to a given row address bit configuration should be read in from this file. Like the params file, if you want to make adjustments to the variables being assigned here, just alter the fscanf command in the ";;READ BITSPEC FILE TO DETERMINE BIT-SPECIFIC PARAMETERS" section. Again, make sure that the second argument of the fscanf command matches the format of the bitspec file.

The final file read in is temp.txt. This file is a temporary file created by SweepRows.pl to hold the values of the desired n- and k-buffers. If a sweep test is desired (Test 1 or Test2), then

these values will both be initialized to -1 and read in as -1. Then, later in the code, the program will check these values to see if they are -1 or not. If they are -1, then the program will know a sweep was desired, and all results will be reported in the output,txt file. Otherwise, if the n- and k-buffers were not read in as -1 (because Test 3 was specified in SweepRows.pl), then only the desired n- and k-buffers combination will be reported in the output.txt file. This file was made a temporary file since it is something that the program can handle during runtime. Once the program finishes, it deletes the temp.txt file (just like it does to the TranResults directory).

In the final section, the calculations and simulation section, the first thing done is to specify the output file (overwrites anything with the name output.txt in the current working directory). This is where "output.txt" gets its name. If you want to change the name of this file, this is where you would do it. Also be sure, however, to change the SweepRows.pl file to reflect the change in name of the output file, so that it knows which file to concatenate to results.txt. Once the output file is set, the program goes through a series of checks to be sure the input parameters are logical. For example, it will check to make sure the word size is appropriate and that the memory size is appropriate. If it is determined that any of these checks do not pass, then the program will stop running the run.ocn file, exit out of ocean, and then send control back to SweepRows.pl. Currently, the program will not completely cease once it realizes that there is a memory size error or word size error. Instead, it will cycle through all of the *bits directories and finish out the rest of the perl script. This is not too much of a problem, since no results will be reported in results.txt (it will just be blank besides the header), but it is a waste of computer operations. In the future, it may be of interest to create a second output file that would allow the SweepRows.pl script to figure out whether a memory or word size error occurred.

Once the program finishes these set of checks, it moves on to check whether the given number of row address bits can support the given size memory (for details on this, consult "Improving the Performance of a CAD Tool Designed to Generate Optimized Virtual Prototypes of Complete SRAM Macros"). If the current row address bit configuration cannot support the memory size, the run.ocn file will stop, the output.txt file will remain empty, ocean will be closed, and SweepRows.pl will continue cycling through the other row address bit configurations. Assuming all checks have passed, the program will calculate the values that need to be passed to the netlist via design variables (number of bitcells per row, parasitic resistance and capacitance of predecode wires). Then, the simulation begins. The current simulation that is run is a 32-nanosecond simulation. Once the simulation is complete, the program will check if SweepRows.pl wanted a sweep of n- and k-buffers or a specific combination, as mentioned earlier. Once this is determined, the code cycles through the various models (for a sweep test) or finds the proper model (for a particular combination) and reports the desired test values. The power value being reported is the average power of the DUT over the 32 nanoseconds. The delay value is the t_{plh} value from the first fall of the input to the first rise of the word line. To change the values that are reported by run.ocn, simply adjust every fprintf command that appears after each DUT is manipulated to produce the delay and power results.

There are a few points to make about the run.ocn that will explain some of its redundancy. First, there is not an easy way to iterate through the different n- and k- buffer combinations, while at the same time being able to tell the simulator which DUT you want (i.e. DUT1, DUT2, DUT3, etc.) because I could not figure out a way to concatenate strings in Ocean. I looked through the file OceanRef.pdf to find a way to do this, but I was unable to do so. If this

were possible, it would reduce a lot of the redundancy in the code, and you could simply create a double-nested loop over n- and k-buffers and use far fewer fprintf commands.

Wrap-Up

After reading this User Guide, you should be able to navigate through the Decoder Optimization Plugin and understand the interdependencies of the code. To make understanding the relationship between the different files easier, I have included a Summary of Interdependencies, as well as a Diagram of Interdependencies. They are on the next pages.

This program can easily be integrated into a technology-agnostic simulation environment (TASE). By following the steps set forth in the TASE_UG on the wiki, this program can function using any process technology. Moreover, once in conjunction with ViPro, you will be able to sweep across VDD values or any other values of interest as well. Since the program currently produces a menu of power-delay results, ViPro will need to select which one of these power-delay combinations "optimizes" a given SRAM.

Summary of Interdependencies

SweepRows.pl

Called/Accessed by:

-Terminal Window using command “perl SweepRows.pl” to begin execution of program

Actions:

-Copies run.ocn into each *bits directory
-Copies header of output_format.txt into results.txt
-Copies contents of output.txt from each *bits directory into results.txt
-Starts Ocean with –nograph option / ocnloader script
-Removes simulation files after completing simulation

Files it Directly Accesses:

-run.ocn
-output_format.txt
-ocnloader
-results.txt
-output.txt
-*bits directories (obviously not a file, but these directories must follow naming convention to be accessed in perl script

output_format.txt

Called/Accessed by:

- SweepRows.pl copies its contents into results.txt

Actions:

-none, it is passive

Files it Directly Accesses:

-none

params

Called/Accessed by:

- run.ocn reads in the parameter values and stores them in variables in the ocean script

Actions:

-none, it is passive

Files it Directly Accesses:

-none

bitspec

Called/Accessed by:

- run.ocn reads in the parameter values and stores them in variables in the ocean script

Actions:

-none, it is passive

Files it Directly Accesses:

-none

output.txt

Called/Accessed by:

- run.ocn creates this file in each *bits directory with the desired output from the simulation
-SweepRows.pl concatenates the contents of each output.txt file into results.txt

Actions:

-none, it is passive

Files it Directly Accesses:

-none

results.txt

Called/Accessed by:

-SweepRows.pl creates this file
-SweepRows.pl copies the contents of output_format.txt into it as the header
-SweepRows.pl concatenates the contents of each *bits directory's output.txt into it

Actions:

-none, it is passive

Files it Directly Accesses:

-none

netlist

Called/Accessed by:

-run.ocn simulates the DUTs in the netlist file

Actions:

-none, it is passive

Files it Directly Accesses:

-none

netlistHeader

Notes: required for netlist to be read into simulator properly

netlistFooter

Notes: required for netlist to be read into simulator properly